Subject: Date:

Data structure

Data structure: is a data orgnization, managment and storage Format That enables efficient access and modification, affects The design.

Classification of DS

- Primitive (Basic)
  - integer
  - Float
  - char
  - Pointer
- Non_Primitive (Advanced)
  - [1] Linear List (arranged)
    - Static Data structure: Array
    - Dynamic Data structure:
      - Linked List
      - stack
      - Queue
  - [2] Non Linear List
    - tree
    - graph
    - Non_Primitive derived From Primitive data structure.

Data

- homogeneus: same type data
- heterogenous: diffrent type data

Speed_Efficiency Program

- Algorithm
  - ✓ time
  - ✗ space
  - } Complexity
  - Focus on Software
- Hardware
  - CPU, Ram, OS
  - Called Performance

time: Amount of time required by an algorithm to executed.

space: Amount of space required by an algorithm

الممسوحة ضوئيا بـ CamScanner

Subject: Date:

Complexity

Search

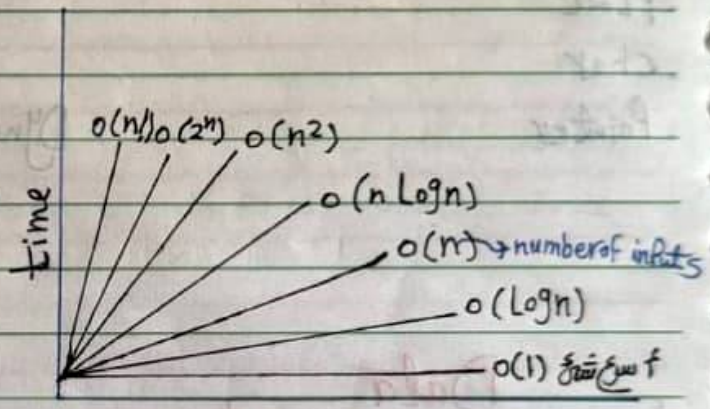
best case averagecase Worst case

best case = omega Notation Ω
average case: theta Notation Θ
Worst case: Big O Notation

- **Big O Notation:** Relation between Input / time scale.
  not tells you the exact time algorithm runtime is. time cabs
  the big_o orgainally stand "order of".
  ~~not affect input.~~

1) **Big o(1): Constant time**
   - Not affect input

* ⊡ *, /, %, ^, +, -, ++, --, +=, -=, /=, *=
  intlization Ex: i=0, else, if, ifelse
  Constant time o(1)



Ex:

```
char LastChar(string s1){
return s1[s1.length()-1];
}
```

1 1 1 1 1

Complexity = o(1)*4 = O(1)
Running time "Exact time" → order "Asymptomatic time"

2) **Big o(n): Linear time Function**
   - affect by input size

Ex:
```
int sum;
For(int i=0; i<n; i++) //n
    sum+=i; //1
```
∴ time Complexity = 1+n = o(n)
// worst case

Note: $o(n^3)$ called Cubic

Ex: $o(n^2)$: Quadric time function

```
int j=0;          //1
For(int i=0; i<n; i++) //n
  For(int j=0; j<n; i++) //n
    Cout<<j<<i;
```

Complexity = 1 + n*n = $o(n^2)$
worst case

Subject: Date:

Note: Log(n) <= * / (loop) إذا كانت الزيادة في

Linear type o(n) <= + -

Note: Loops: order of body * iteration

[3] Big o(Logn): Logarithmic

Ex:

```
For(int i=0; i<n; i=i*2)
```

Complexity = $O(Log_2 n)$

Ex: while(n>2){

n=n/4;

}

Complexity = $O(Log_4 n)$

[4] n Logn Big o(nlogn): Log Linear

Ex:

```
For(int i=0; i<n; i++) //n
  For(int j=0; j<n; j=j/3)//Logn
```

∴ Complexity = $n * Log_3 n = O(n Log_3 n)$

Ex:

```
for(int i=n/2; i<n; i++) //n/2 ≃ n
  For(int j=0; j<n; j*=2) //Logn
    For(int k=0; k<n; k/=2) //Logn
```

Complexity = $n * Log_2 n^2 = o(n Log_2 n^2)$

Subject: Date:

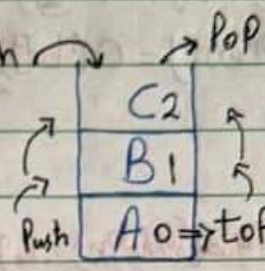
Stack

- Using Array or Linked List (Logical)

1) Stack Using Array

0	1	2	3

Array



- LiFo: Last in First out Stack العيدة اللي ماشي عليه
- FiLo: First in Last out " " " " "
- آخر واحد أضاف هو أول واحد هيخرج ← عدد العناصر = top+1

Ex:

```
Class stack{
Private:                    ← to access the last item in stack
        int size, top;
        int item[ ]; // Array to store stack Element
Public:
```

① stack(int size)
{
This size=size;
top=-1;
}

② Void Push(int x)
{
if(top>=size-1) ← Full
Cout << "Stack is full";
else{
item[++top]= top++;
item[top]=x;
}

③ bool isEmpty(){
Return top<0;
}

④ Void Pop(){
if(isEmpty())
Cout << "Stack Empty"
else
top--;

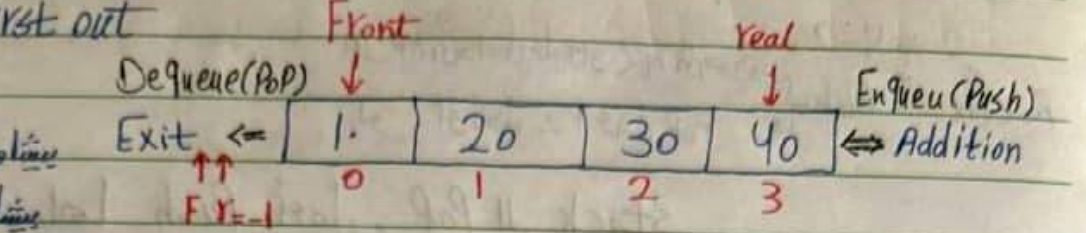
⑤ Void getTop(int &stack top) ← عشان تغير القيمة
if(IsEmpty())
Cout << "Stack Empty";
else{
stacktop=item[top];
Cout << stacktop;

⑥ Void Print(){
For(int i=top; i>=0; i--)
Cout << item[i];

Subject: Date:

Simple Queue

- FiFo: First in first out



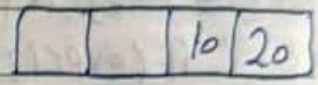
Front (First): بيشاور أول واحد
Rear (end): بيشاور آخر واحد

F R = -1

- size = 4 Rear = 3 ∴ Rear == size - 1 ⇒ Full
- if initLization rear, Front = -1 ∴ empty when Front, rear = -1
- if Front == rear ∴ there is one item لازم بعدها ترجع empty queue

- Disadvantage

هكذا يعتبر الـ Queue الأماكن الخالية مقدرش أرجع أملاها تاني مقدرش أستفاد منها ⇐ Full



سيتم حلها Circular Queue

Ex: Class simple Queue {

Private:

int Front, rear, size, *Queue

Public:

1) simple Queue (int _size)
   _size = size;
   Front = rear = -1;
   Queue = new int (_size);

2) bool isEmpty()
   return (rear == -1 && Front == -1);

3) bool isFull()
   return (rear == size - 1);

Print
For (Front → rear)

4) int Front()
   Rear ()
   if (isEmpty())
   return -1;
   else
   return Queue [Front];
   rear

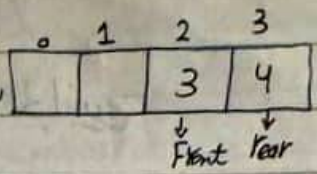
5) Void Equeue (int x)
   if (isFull()) Cout << "Full";
   else if (isEmpty()) F = r = 0;
   else
   rear++;
   Queue [rear] = x;

6) int Dequeue()
   if (isEmpty()) return -1;
   else if (Front == rear)
   1) int x = Queue [Front]
   Front = rear = -1;
   2) return x;
   else
   1) Front++; 2)

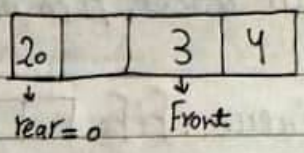
Subject: Date:

Circular Queue

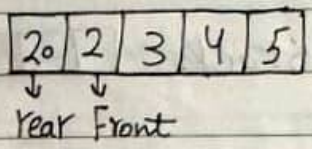
- The Queue is full to solve this Problem Using Queue "Circular"



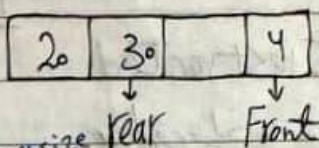
- to Enqueu number when full ⇒ (Rear+1) % size = 4%4=0



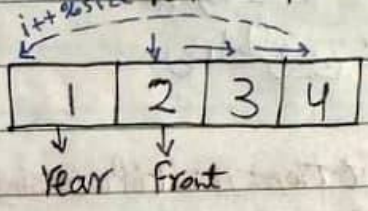
- if (Rear+1) % size == Front ∴ is full



- to Dequeu (Front+1) % size to Return Zero



- When Print Queue Circular (i+1) % size



- Changed Function

- bool isFull()

```
return (rear+1)%size==front;
```

- int Dequeue()

```
if (isEmpty()) return -1;
else if (front==rear)
    int x=Queue[front];
    front=rear=-1;
else
    int x=... front=(front+1)%size
...
return x;
```

- Void Enqueue (int x)

```
if (isFull())
    Cout<<"full"; return false;
else if (isEmpty()) front=rear=0;
else
    rear=(rear+1)%size;
...
Queue[rear]=x; return 1;
```

- Void Print()

```
For (int i=front; i!=rear; (i+1)%size
    Cout<<Queue[i];
...
Cout<<Queue[rear];
```

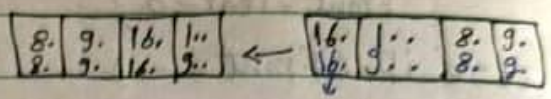
الممسوحة ضوئيا بـ CamScanner

Subject: Date:

Priority Queue

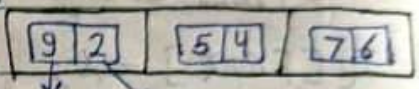
عبارة عن رقم هيتحدد ال Priority بتاعه, كل رقم هيخش هيبقى معه FiFo فيه Queue لأ كنها لا تتبع مبدأ FiFo

بترتب على أصغر Priority, أصغر رقم ال Pro هو هيكون أول واحد



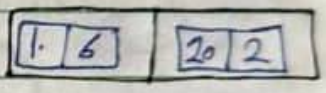

```
Struct Item{
   int Value, Priority
}
```

كل عنصر في Q هتخزن جواه Struct




```
Class Priority{
    int Front, rear, size;
    Item *Q;
}
```

لازم ترتيبها



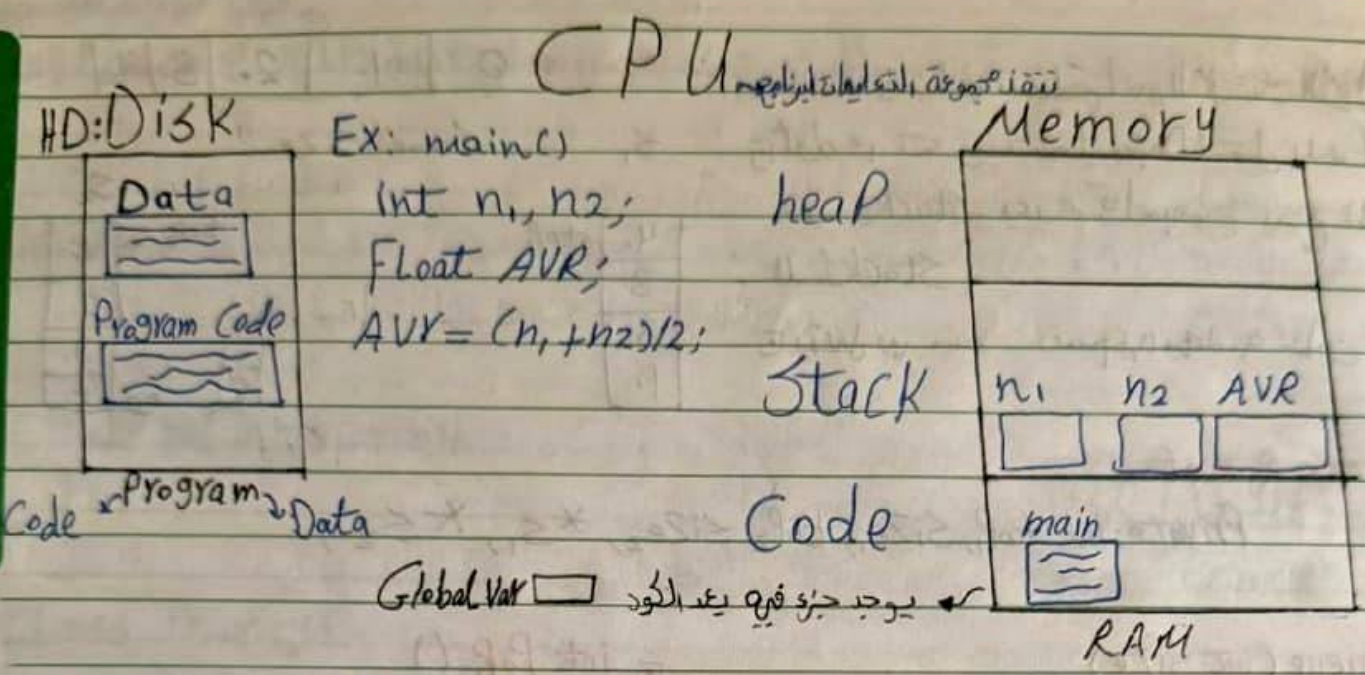
```
int DeQueue()
   =
int X=Q[Front].Value;
```

```
bool enqueu(int x, int P)
 if(isFull()) cout<<"Full"; return False;
else if(isEmpty())
rear=Front=0; Q[rear].Val=X; Q[rear].Pr=P;
else
   rear=(rear+1)%size
 Q[rear].Val=x;
 Q[rear].Pro=P;
   insertionSort();
 return true;
```

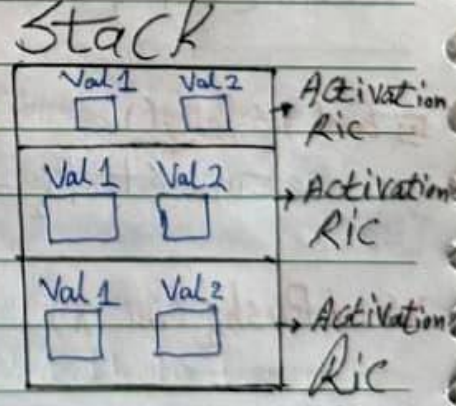
```
Void insertionSort()
   int j;
    Item temP;
For(int i=0; i<=rear; i++){
    j=i-1;
    temP.Val=Q[i].Val;
    temP.Pro=Q[i].Pro;
While(j>=0 && temP.Pro<Q[j].Pro]
Val, Pro ← Q[j+1]=Q[j];
      j--;}
   Q[j+1]=temP;
}
```

Subject: Date:

Static Memory Allocation



- يتحجز مكان في ال Memory عشان data, Value ال Memory عشان تخزن فيها
- وهذا يسمى Memory allocat ← Stack تخزن في ال Stack ال Primitive ال Size
- بتاعها معروف مسبقاً وهذا Compile time ← الوقت اللي قبل م ال Prog ينفذ.
- يتم حساب مساحة البرنامج أثناء إجراء عملية الترجمة ولا تتغير هذه القيمة Compile time



- من الممكن أن يكون دوال أخرى غير ال main لهذا تقسم كل واحدة وكل واحدة تسمى Activation Ricords (Ric)

- Static Array Memory
  - Memory is allocated at Compile-time
  - Declaration of Large memory variable must be determind before the Program is Compiled, Ex: int Student[10]; ✓
    if we don't no exact number using Dynamic

Subject: Date:

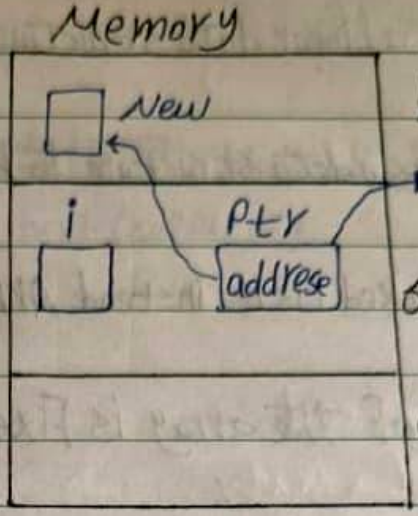
Dynamic Memory Allocation

Ex:

```
int i;
Float * Ptr;
Cin >> i; //size array
Ptr = new Float [i];
```

يتحجز مساحة في الـ Memory في الـ heap

```
delete [] Ptr;
return 0;
```



- الـ size مبيبقاش معرف مسبقاً "بيتوقف على حاجات كتيرة أثناء Runtime"
- Run time: بيتم عمل طلب ذاكرة إضافية أثناء التشغيل ويمكن حذفها عندما لم تعد هناك حاجة لها
- أي مساحة أنت حجزتها في الـ heap لازم تعملها delete عشان متظلش المساحة محجوزة بدون فائدة
- لو أنت مبتعملش delete ممكن مرة في مرة الـ heap تمتلئ Memory فلما تشغل الـ Program آخر يقولك Memory full

Dynamic Memory

=> We Can "ask" For Memory While The Program is Running.
=> We Can "allocate" and "release" memory ourselves "Dynamicly.

Subject:

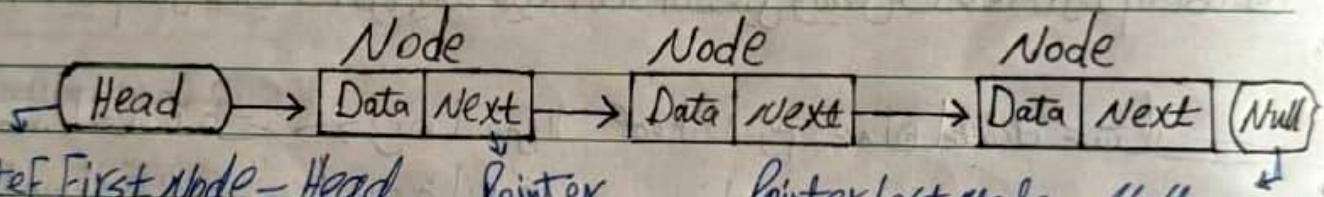
Linked List

- Linked List is a Linear data structure that encluded a series of Connected Node
- Linked List is a data structure to build other data structure Like Queu, stack...
- Why do use Linked List instead Array ?

1) The size of the array is Fixed Acess time: o(1)

2) insertion, Deleted a new Element is also expensive with Array

⇒ Normal / SingLy Linked List



1) Dynamic Size, Runtime

2) Easy of insertion, Deleted

- Disadvantage

1) Random Acess is not allowed

متقدرش تصل مباشرةً لـ Node معينة لازم تمر على كل الـ Node اللي قبل عشان توصلها (صعب النزول للآخر) List

2) Extra Memory Space For Pointer is required with each Elemt.

Subject: Date:

Singly Linked List implmentation

```
struct Node {
    int data;
    Node *next;
    Node(int& x) : data(x), next(nullPtr) {}
};
```

Pointer هيشاور على الـ Node [Pointer] [D|N]

```
class SLL {
    Node * head;
    siz_t size_L;

Public:
    SSL(): head(NULL), size_L(0){ }
    bool isEmpty(){ return (head == null)}
```

Linked List Empty

```
Void insertFirst(int x){
    if(isempty())
    {
    Node* Newnode = new Node();
    newnode->data = x;
    newnode->next = NULL;
     head = newnode;
    }
else {
    Node* Newnode = new Node();
    Newnode->data = x;
    Newnode->next = head;
     head = Newnode; }
```

Case 1 insert-First

لو عايز تعيش وهو فاضي: [head] → NULL

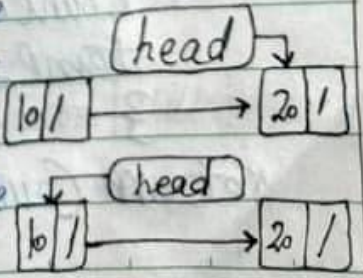
1. Creat New node [data|next]
2. Fill data Part
3. Set Next Pointer = Null
4. Head Points to Newnode

[head] → [D|N]

Case 2

items لو الـ List بها [head] → [20|/]

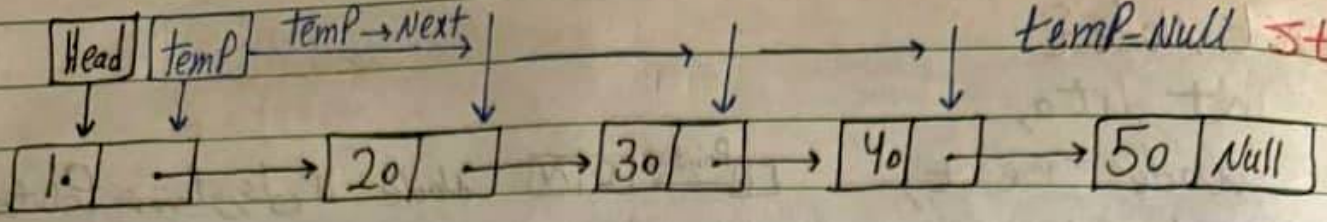
1. Creat New node
2. Fill data Part
3. Set Next → Head
4. Set Head Points to Node



الممسوحة ضوئيا بـ CamScanner

Subject:

Traversing



مينفعش تدخل على الـ List مباشرة من أي Node لازم تعدي من أول Head إلى الآخر

Node *temP=head; ← لازم نعمل temP ومينفعش نستخدم Head اللي لو استخدمت
temP=temP→Next; Head هتخليه يشاور على أول Node وكده
كل الـ items اللي في الـ List ضاعت في الـ Memory

∴ شرط التوقف temP=Null

⇒ Using While instead of For Because I don't Know Num of Node

```
Void Print(){
    Node *temP=head;
    While(temP!=Null){
        Cout<<temP→data;
// temP=temP→Next;}
```
ينتقل للـ Node التانية

```
int Count(){
  int Counter=0;
  Node *temP=head;
  While (temP!=null){
      Counter++;
      temP=temP→Next;
    }
return Counter;
```

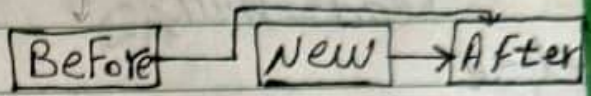
```
bool isFound(int key){
bool Found=False;
Node *temP=head;
While(temP!=null){
  iF (temP→data==key)
t→next; Found=true;}
  return Found;
```

Subject: Date:

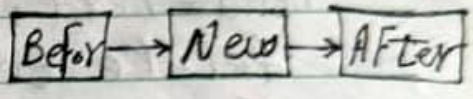
Insert Position

```
Void InsertPos(int X, int Pos){
 iF(isEmpty())
     insertFirst(X);
 iF(isFound(Pos)){
   Node* Newnode = new Node();
    Newnode->data= X;
    Node * temp=head;
While(temp!=null && temp->next->data!=Pos)
     temp=temp->next;}
  newnode->next= temp->next;
   temp->next = newnode;}
else{cout "Item Not Found";}
```

1. Creat a New Node
2. Fill data Part
3. traverse Before the item they want
4. Set the New node Next Pointer to the Current nod's Next



5. Update the Current nod's Next to the newnode.



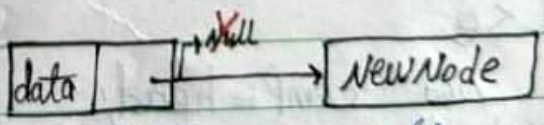
Note: While (temp==null) out of List

Insert at End

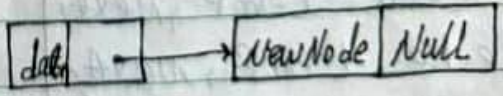
```
Void aPPend(int X){
 iF(is empty())
     insertfirst(X);
else{
   Node * temp=head;
 While(temp->Next!=null){
     temp=temp->Next;}
 Node * newnode=new Node();
    temp->Next=newnode;
   newnode->Next=null
   }
}
```

1. Creat New Node
2. Fill data Part
3. traverse to The End of List

   While (temp->Next!=null)
4. Set the Last Node Next Pointer to the newnode



5. New Node Points to Null



الممسوحة ضوئيا بـ CamScanner

Subject: Date:

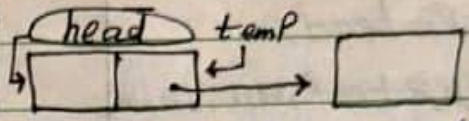
Delete

Delete First

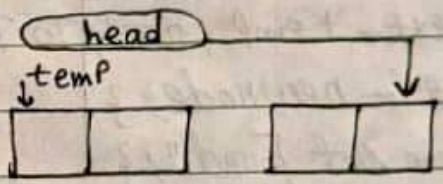
```
Void deleteFirst (){
 iF (isempty())
 Cout << "Linked list is Empty";
iF (head->Next == Null)
    delete head;
     head = null;
else
   Nod * temp = head;
    head = head->next;
     delete temp;
     }
```

1] iF Linked List with at 1 item
head->Next == Null
① delete head ② head = NULL

2] Using temp Pointer to help deleting



3] Head Points Head->Next

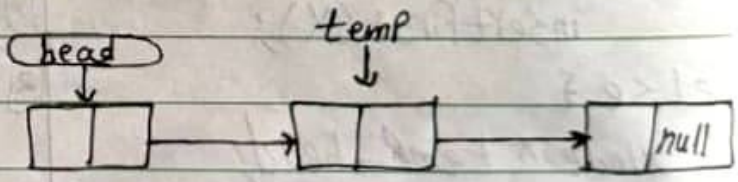


head = head->Next
delete temp

Delete End

```
Void deleteEnd (){
 iF(isempty())
  Cout << "List is Empty";
iF(head->Next == null)
    delete head;
     head = null;
else
    Nod * temp = head;
While (temp->Next->Next != null)
      temp = temp->Next;
 delete temp->Next;
   temp->Next = null;
```

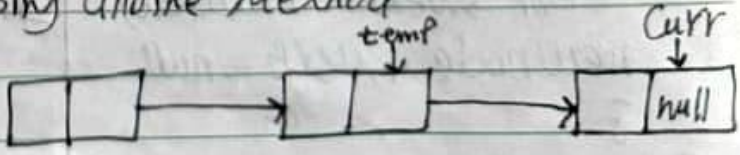
1] Using temp to Connect Penultimate Node



While (temp->Next->Next != null)

2] delete temp->Next && temp->Next = null

Note:
=> Using anothe Method



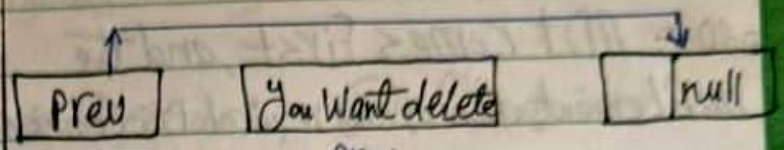
الممسوحة ضوئيا بـ CamScanner

Subject: Date:

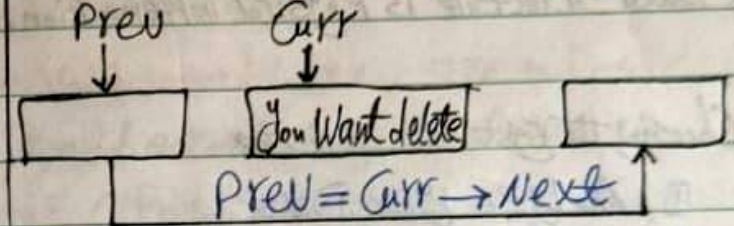
Delete Position

```
Void deleteSpecific(int item){
  if(isempty())
Cout<<"List is Empty";
if(head->data==item)
   deleteFirst();
if(head->Next==null)
      delete head;
      head=null;
else
Node* Prev=null;
Node* Curr= head;
While (Curr->data!=item){
      Prev=Curr;
     Curr= Curr->Next;
   }
Prev->Next=Curr->Next;
   delete Curr;
Note: After While adding  if (Curr==null) Cout<<"Not Found";
```

1] Set Prev Node Points to Next After delete



2] Using 2 Ptr → Prev
→ Curr: Points to Node delete



3] if You want delete First item
(head->data==item)
∴ Using Fn Delete at First

الممسوحة ضوئيا بـ CamScanner

Subject: Date:

Abstract Data type

- We can build any Data structure with a specific characteristic and operations.
- ADT: ADT comes First, and the implementation of the required DS comes Next.

ADT

name DS
Characteristic DS
operation

⇒ Data structure is Physical implemention of an ADT

- Choosing the right Data_structure
  1. Analyze your Data
  2. identify operations
  3. time Complexity
  4. Scalability (Potential growth of your Data)

⇒ Physical Data structure: Array : مثل DS أخرى عند تنفيذها لا تعتمد على

⇒ Logical Data Structure: Static, Queue: مثل DS أخرى في تنفيذها تعتمد على

Note: Data type size_t: Float, int, Char

Subject: Date:

Static Array

- Array is a basic Data structure store Fixed_size Collection element Same type
- these elements arranged in Contiguous memory Location

- Fixed size memory allocated in Stack Compile time
- Array is Homogenus Element

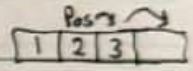
- Void APPend (t &V)
  if (size_a >= Capacity)
    throw out_of_range";
  a[size_a++]=V;
  // add in the Last Element

- t RemoVe () //deLeTe LasTelemnt
  if (size_a==0)
    throw ___;
  Yetern a[__size_a];

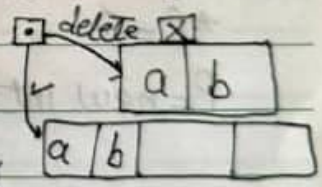
- Void Delete (size_t Pos)
  // delete according Position
  if (size_a==0)
    (Pos >= size_a)

  For (size_t i=Pos ; i<size_a ; i++)
    a[i]=a[i+1]; //Shift to Left
  size_a___;

- Void insert(t &x , size_t Pos)
  // add according to Position
  if (Pos >= Capacity)
    (Pos > size_a)
    (size_a >= Capacity) throw
  For (size_t i=size_a; i>Pos ; i--)
    a[i]=a[i-1]; //Shift to right
  size_a++;
  a[Pos]=x;



- // ReSize when Capacity full in Dynamic
  Void resize (C)
  // Allocate New Memory
  t* temp=newt[C];
  // Copy oLd Array to new Array
  For(size_t i=0; i<size_a ; i++)
    temp[i]=a[i];
  // Free old Memory, Update Pointer
  delete []a;
  a=temp;
  Capacity=C;



Dynamic ويكون resize (Capa*2)

resize بـ APPend و throw بدل insert ← ملوا اول

الممسوحة ضوئيا بـ CamScanner

Subject: Date:

DYnamic Array

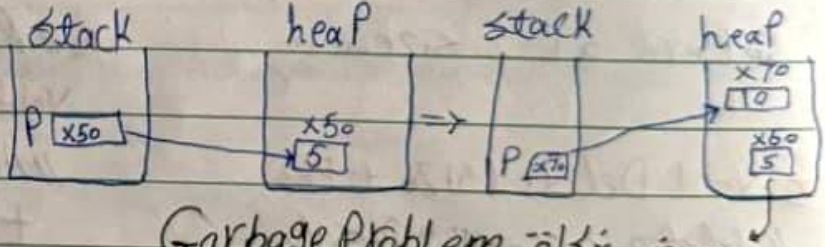
Key Words: new (allocation)

- Const int s=1000000;
  int a[s]; → to solve this DMA
  Error: stack overFlow
- int *a = new int [1000000];
  ↑ Pointer يستقبلوا في ال ← addresse يرجع

⇒ Dynamic Array modified during Program runtime.
⇒ Dynamic Array element occupy a contiguous block of Memory.
⇒ static Array allocated a predetermind amount of Memory and size can not change
⇒ Dynamic can expand size grow its Memory size by a certain Factor.

Ex: int *P=new int (10);
- ASK the OS to Find a 4byte Memory
- the os will search for a free Memory in heap
- the os will return addrese

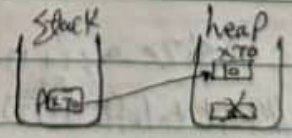
- int *P=new int;
  *P=5
  P=new int (10);



Garbage Problem مشكلة حدث

- Garbage_Memory Leak Problem: Make sure to delete Memory when Finish

∴ int *P=new int;
*P=5
delete P; → لا تحذف P تحذف الذاكرة التي يشاور عليها P
P=new int (10);



الممسوحة ضوئيا بـ CamScanner

Subject: Date:

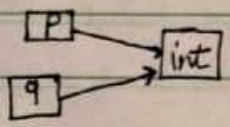
delete: Keyword dos'nt delete the Pointer itself it delete what the Pointer refers to.

→ to delete Pointer make it Point to nothing null Ex: P=null;

Ex: //unitiLization Djnamic Array
int *d=new int[500]; // Vandom Value
// initilized to zero
int *d=new Int[500](); //all Value=Zero
delete []d; // Freeing, Releasing, deleting

Dangling Pointer Problem

```
int *p, *q;
p=new int;
q=p
delete p;
```



بيشاور على مكان مش موجود

Static Note

- Memory is acquired automatic
- Memory is returned automatic when object goes out of Scope.

delete أول ما البرنامج في static بخلص } بيعمل automatic delete وشيء محتاج

الممسوحة ضوئيا بـ CamScanner